Année 2009~2010

Architecture des Systèmes
EPITA
InfoSPE

DUJARDIN Anne-Sophie

Sommaire

Chap.1. Les nombres à virgule flottante

I. Introduction

1. Définition

Les nombres à virgule flottante sont des approximations rationnelles des nombres réels. C'est, d'ailleurs, une des représentations les plus utilisées dans les systèmes informatiques.

Les nombres à virgules flottante possèdent un signe s, une mantisse entière m (parfois aussi appelée significande) et un exposant e. Un tel triplet représente un réel s.m.be, où b est la base de représentation (souvent 2 mais aussi 16). En faisant varier e, on fait « flotter » la virgule. m est généralement de taille fixée. Cette notion s'oppose à la représentation dite en virgule fixe où c'est l'exposant e qui est fixé.

Les différences de représentation interne des nombres flottants d'un ordinateur à un autre obligeait à reprendre finement les programmes de calculs scientifiques pour les porter d'une machine à une autre jusqu'à ce qu'un format normalisé soit proposé par l'Institute of Electrical and Electronics Engineers (IEEE).

2. Norme IEEE 754

La norme IEEE *Standard for binary Floating-Point Arithmetic* (ANSI/IEEE Standard 754 - 1985) a été définie dans le but d'améliorer la qualité du calcul flottant et la portabilité des applications. Ce standard est maintenant utilisé et respecté par tous les acteurs principaux du calcul scientifique. Ainsi, la quasi-totalité des architectures actuelles incluent une implémentation matérielle des calculs sur les flottants IEEE, directement dans le microprocesseur, garantissant une exécution rapide.

Cette norme spécifie 2 formats de nombre en virgule flottant (aussi appelés flottants) et les opérations associées. Ces flottants peuvent être codés sur 32 bits (flottants simple précision) ou 64 bits (double précision). Ils sont définis par 3 champs : un signe S, une mantisse M et un exposant E. Un nombre ainsi codé est donné par :

$$(-1)^S . (1, M)_2 . 2^{E-Biais}$$

La répartition des bits et la valeur du biais pour chacun des formats est donnée par le tableau suivant :

	Signe(S)	Exposant (E)	Mantisse (M)	Biais
Simple Précision	1	8	23	127
Double Précision	1	11	52	1023

II. Conversions

1. Décimal -> IEEE 754

Pour convertir un nombre réel r en flottant IEEE, il faut :

- ✓ Déterminer le bit de signe S :
  - si $r > 0, S = 0$
  - si $r < 0, S = 1$
- ✓ Convertir la valeur absolue en binaire fractionnaire avec le maximum de précision possible selon le format utilisé

Rappel : Conversion d'un réel en binaire fractionnaire :
- Partie entière : méthode des divisions successives par 2
- Partie Fractionnaire : méthode des multiplications successives par 2

Pour plus de précision, se reporter au cours de SUP.

- ✓ Normaliser le nombre binaire fractionnaire obtenu, c'est-à-dire le mettre sous la forme $(1, \mathrm{M}).2^{\mathrm{e}}$
- ✓ Déterminer E en fonction de e et du biais
- ✓ Ecrire le résultat sous forme binaire en juxtaposant les différents champs de la façon suivante : SEM

2. IEEE 754 -> Décimal

Pour convertir un flottant IEEE en décimal, il faut :

- ✓ Déterminer le signe en fonction de S
- ✓ Déterminer e en fonction de E et du Biais
- ✓ Déterminer $\mathrm{m}_2 = (1, \mathrm{M})_2$ sous forme binaire fractionnaire
- ✓ Ecrire le résultat sous la forme $\pm \mathrm{m}_2.2^{\mathrm{e}}$
- ✓ Simplifier si nécessaire (en déplaçant la virgule suivant la puissance de 2) et convertir le nombre binaire obtenu en décimal (cf. cours de SUP)

3. Représentation des valeurs particulières

a. Le Zéro

Pour coder la valeur décimale 0, les champs M et E sont à 0. Seul S peut (éventuellement) être à 1

b. Les Infinis

- ✓ $S =$ Signe de l'Infini
- ✓ E = Que des 1
- ✓ $M = 0$

c. Formats Dénormalisés

Le plus petit nombre, en valeur absolue, admettant une représentation dite normalisée en format IEEE 754 est $2^{1-\text{Biais}}$ (puisque $2^{-\text{Biais}} \leftrightarrow 0$). Or, la norme IEEE 754 permet également de représenter des nombre plus petits en acceptant que leur représentation ne soit pas normalisée.

Une représentation dénormalisée se reconnait à :

$E = 0$ et $M \neq 0$

La valeur ainsi représentée est alors donnée par l'expression suivante : $(-1)^S . m_2 . 2^{1-\text{Biais}}$. La conversion se fait donc quasiment de la même manière qu'en format normalisé.

d. Not A Number

Certains codes sont prévus pour représenter les « erreurs » ou n'importe quelle entité qui ne serait pas un nombre. Ce sont les Not A Number (NaN).

Les codes réservés à cet effet sont de la forme :

$E =$ Que des 1 et $M \neq 0$

Les NaN servent, par exemple, à propager convenablement les erreurs lors des calculs.

4. Synthèse

E	M	Correspondance
0	0	0
0	$\neq 0$	$(-1)^S . m_2 . 2^{1-Biais}$
$\neq 0$ mais pas que des 1	quelconque	$(-1)^S . (1, M)_2 . 2^{E-Biais}$
Que des 1	0	$(-1)^S . \infty$
Que des 1	$\neq 0$	NaN

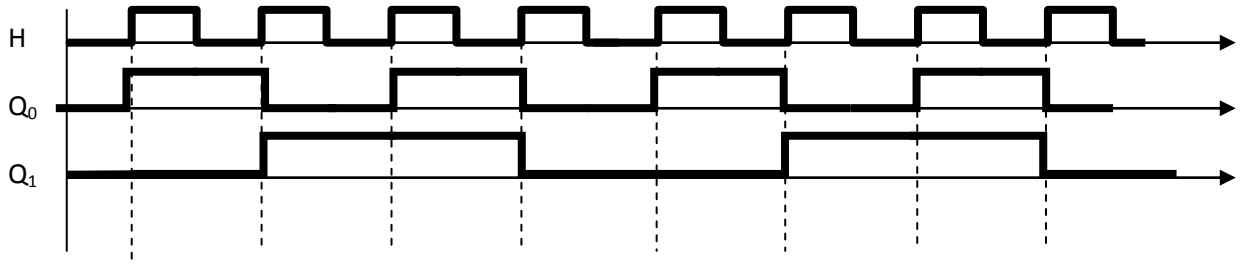
Chap.2. Les Compteurs

I. Principe

Un compteur est un circuit séquentiel qui permet de dénombrer des impulsions appliquées sur son entrée d'horloge (clk ou H) et de restituer sur ses sorties des informations sous forme binaire. A chaque impulsion l'état du compteur est modifié et entre deux impulsions son état reste stable.

Ex : On veut compter 0,1,2,3. On a donc besoin de 2 variables Q1 et Q0.

H	Q_1	Q_0
1^{er} front	0	0
$2^{ème}$ front	0	1
$3^{ème}$ front	1	0
$4^{ème}$ front	1	1
$5^{ème}$ front	0	0



Il y a deux types de comptage :

- ✓ Le comptage asynchrone :
  Il s'agit d'une association de bascules en série. Chaque bascule fait commuter la suivante par liaison sortie~horloge. Chaque bascule possède donc une horloge différente.
- ✓ Le comptage synchrone :
  Il s'agit d'une association de bascules en parallèle. Chaque bascule commute en fonction de l'état des sorties précédentes. Ici, toutes les bascules ont la même horloge.

Il y a deux types de cycles :

- ✓ Les cycles complets:
  Le compteur passe par tous les 2^n états définis par les sorties des n bascules.
- ✓ Le comptage synchrone :
  Le compteur parcourt une partie seulement des 2^n états définis par les sorties des n bascules.

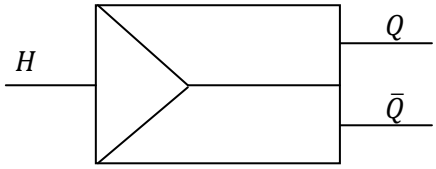
Dans tous le reste du cours, on n'utilisera des bascules D ou JK synchronisées sur front (montant ou descendant).

II. Les compteurs/décompteurs asynchrones

1. Bascules câblées en commutation permanente

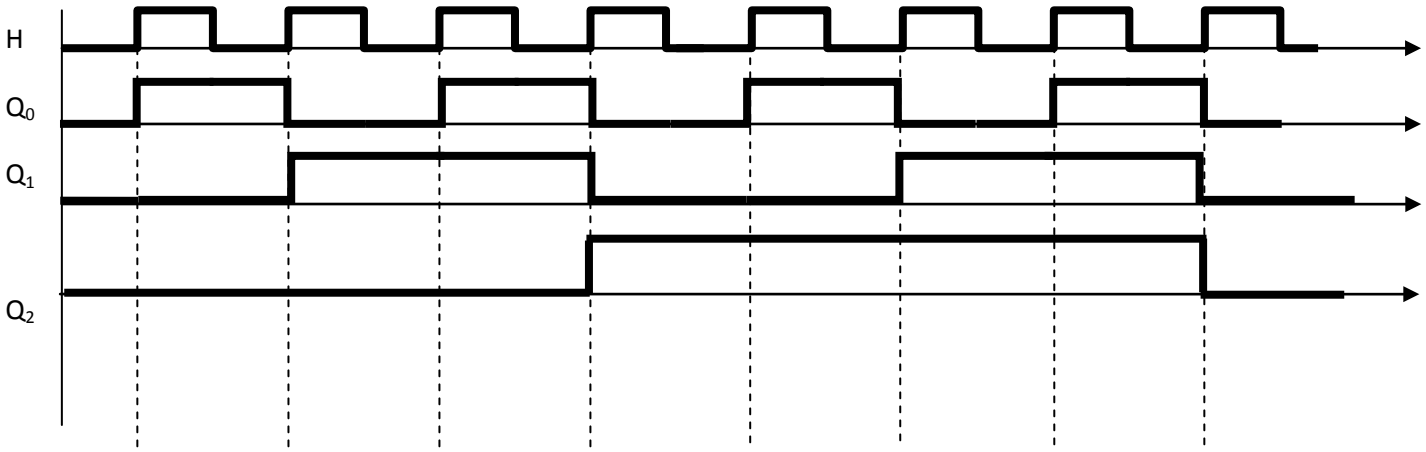
Une bascule câblée en commutation permanente change d'état à chaque niveau actif d'horloge.

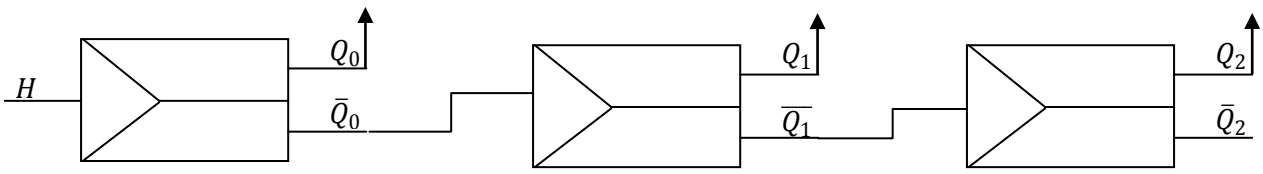
Notation générale : Bascule D ou JK câblée en commutation permanente :



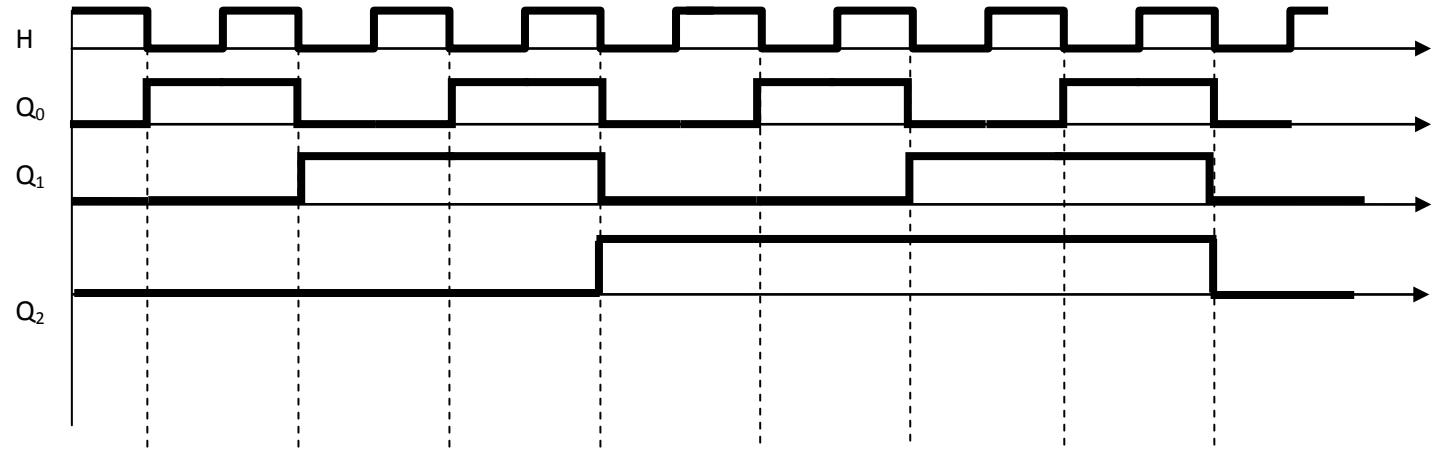
2. Comptage asynchrone à cycle complet : Exemple avec 3 bascules

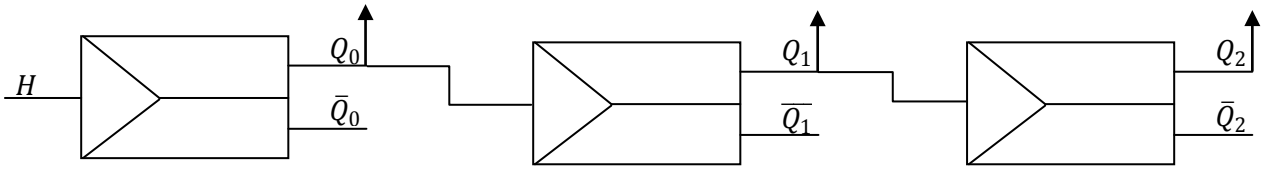
a. Avec bascules synchronisées sur front montant





b. Avec bascules synchronisées sur front descendant





3. Comptage/Décomptage asynchrone à cycle complet : Synthèse

a. Bascules synchronisées sur front montant

- ✓ Liaison Q_i aux H_{i+1} : Décomptage sur les Q_i
  Comptage sur les $\overline{Q_i}$
- ✓ Liaison $\overline{Q_i}$ aux H_{i+1} : Comptage sur les Q_i
  Décomptage sur les $\overline{Q_i}$

b. Bascules synchronisées sur front descendant

- ✓ Liaison Q_i aux H_{i+1} : Décomptage sur les Q_i
  Comptage sur les $\overline{Q_i}$
- ✓ Liaison $\overline{Q_i}$ aux H_{i+1} : Comptage sur les Q_i
  Décomptage sur les $\overline{Q_i}$

4. Comptage/Décomptage asynchrone à cycle incomplet

Exemple : Comptage de 0 à 4 avec des bascules synchronisées sur front descendant.

On veut réaliser le cycle suivant :

On va partir de la base d'un compteur asynchrone à cycle complet et on va agir sur les entrées asynchrones de forçage à 0 et à 1.

On remarque que le compteur compte dans l'ordre jusqu'à 4 et que chaque état doit durer pendant une période d'horloge. Il faut ensuite le remettre à 0.

A quel moment faut-il faire ce forçage à 0 ? Si on force à 0 lorsque le compteur est à 4, l'état 4 ne durera pas pendant une période d'horloge, les entrées de forçage étant asynchrones… Il faut donc laisser passer le compteur à 5 et c'est seulement à ce moment là qu'on doit agir sur les entrées Set et Reset pour remettre le compteur à 0.

Pour cet exemple, supposons les entrées Set et Reset actives au niveau haut.

Q_2	Q_1	Q_0	R_2, R_2, R_0
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	X
1	1	1	X

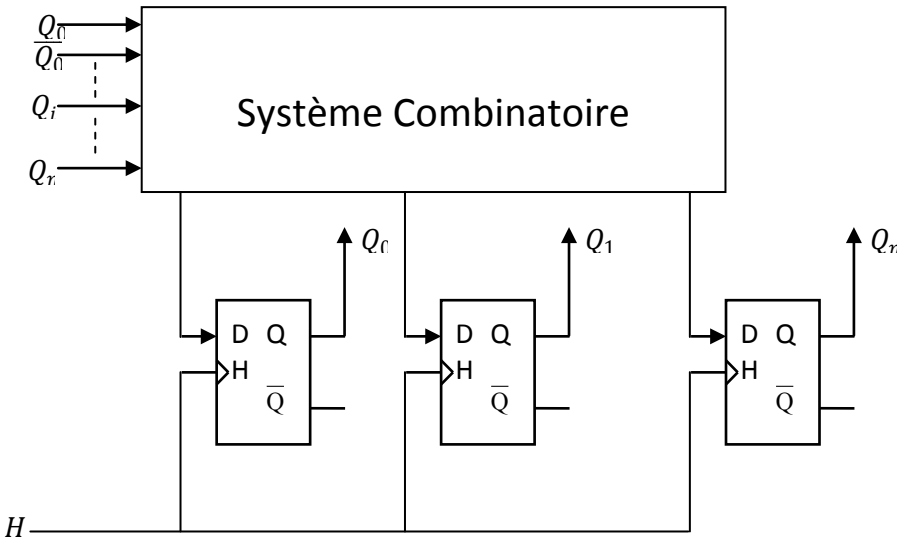
Traduisons cette table de vérité en tableau de Karnaugh :

$Q_2 \backslash Q_1 Q_0$	00	01	11	10
0	1	1	1	1
1	1	0	X	X

On trouve alors : $R_2 = R_2 = R_0 = \overline{Q_2} + \overline{Q_0} = \overline{Q_2 . Q_0}$

III. Compteurs / Décompteurs synchrones

1. Principe



But : Déterminer le système combinatoire qui définit les entrées synchrones pour obtenir le cycle désiré.

2. Réalisation avec des bascules D.

Exemple : Compteur Modulo 8.

Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

La table de vérité ci-dessus permet de déterminer les équations des entrées D des bascules afin de décrire le cycle désiré. Cherchons maintenant les équations.

Solution évidente : $\mathrm{D}_0 = \overline{\mathrm{Q}_0}$

$Q_2\backslash Q_1Q_0$	00	01	11	10
0	0	1	0	1
1	0	1	0	1

$$D_1 = \overline{Q_1}Q_0 + Q_1\overline{Q_0} = Q_1 \oplus Q_0$$

$Q_2\backslash Q_1Q_0$	00	01	11	10
0	0	0	1	0
1	1	1	0	1

$$D_2 = \overline{Q_2}.Q_1Q_0 + Q_2\overline{Q_1} + Q_2\overline{Q_0}$$
$$= \overline{Q_2}Q_1Q_0 + Q_2(\overline{Q_1} + \overline{Q_0})$$
$$= \overline{Q_2}Q_1Q_0 + Q_2(\overline{Q_1.Q_0})$$
$$= Q_2 \oplus (Q_1Q_0)$$

Le principe reste le même pour construire un compteur à cycle incomplet.

3. Réalisation avec des bascules JK.

a. Table des transitions :

Afin de déterminer les équations des entrées J et K de ces bascules, établissons la table des transitions

1 front d'horloge

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

1ière ligne : Q passe de 0 à 0 → état mémoire : J = 0, K = 0
→ mise à 0 : J = 0, K = 1

2ième ligne : Q passe de 0 à 1 → Mise à 1 : J = 1 et K = 0
→ Commutation : J = 1 et K = 1

3ième ligne : Q passe de 1 à 0 → mise à 0 : J = 0 et K = 1
→ Commutation J = 1 et K = 1

4ième ligne : Q passe de 1 à 1 → Etat mémoire: J = 0 et K = 0
→ Commutation J = 1 et K = 0

b. Compteurs à cycles complets et incomplets

Exemple : Compteur Modulo 8.

1 front

Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	X	0	X	1	X
0	0	1	0	X	1	X	X	1
0	1	0	0	X	X	0	1	X
0	1	1	1	X	X	1	X	1
1	0	0	X	0	0	X	1	X
1	0	1	X	0	1	X	X	1
1	1	0	X	0	X	0	1	X
1	1	1	X	1	X	1	X	1

La table de vérité ci-dessus permet de déterminer les équations des entrées D des bascules afin de décrire le cycle désiré. Cherchons maintenant les équations.

Solution évidentes

$$j_0 = k_0 = 1$$
$$J_1 = K_1 = Q_0$$

Pour J_2

$Q_2\backslash Q_1Q_0$	00	01	11	10
0	0	0	1	0
1	X	X	X	X

Pour K_2

$Q_2\backslash Q_1Q_0$	00	01	11	10
0	X	X	X	X
1	0	0	1	0

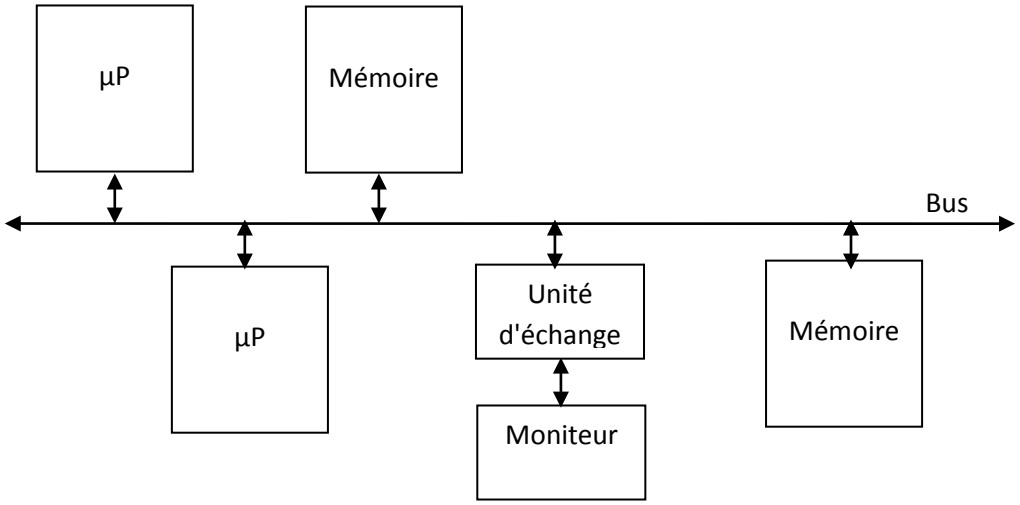
On obtient alors : $J_2 = K_2 = Q_1Q_0$

Le principe reste le même pour construire un compteur à cycle incomplet.

Chap.3. Les Systèmes Numériques

I. Introduction : Définition

Un système numérique est un ensemble de composants numériques autonome comportant au moins un microprocesseur (*µP*).



Le rôle du microprocesseur est de gérer l'ensemble. Il communique avec les autres composants soit directement via le bus, soit par le biais de l'unité d'échange (traducteur de protocole)

II. Les mémoires : Constitution de plans mémoire

1. Introduction

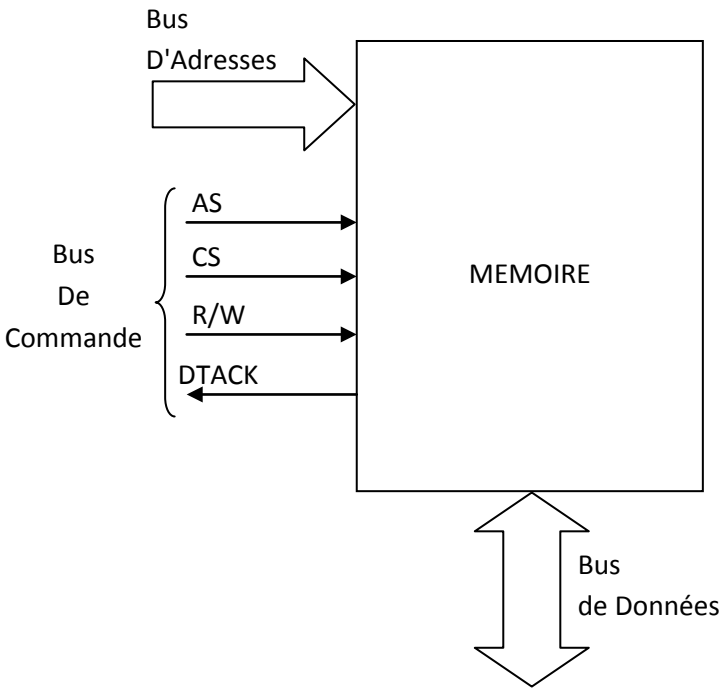
Une mémoire est un composant numérique capable de stocker des informations (instruction et/ou données).

Les systèmes utilisent souvent l'architecture de Von Neumann[1] : la mémoire contient donc à la fois les instructions ET les données.

[1] Rq : Par opposition, l'architecture de type Harvard est une conception de microprocesseurs qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chacune des deux mémoires s'effectue via deux bus distincts. C'est le cas des microcontrôleurs PIC, par exemple.

Les instructions sont donc exécutées de façon séquentielles et dans l'ordre de stockage. Il n'y a donc pas de calculs complexes à faire pour connaître l'adresse de la prochaine instruction.

2. Organisation externe



- ✓ Bus de données : il contient la donnée à lire ou à écrire. Il est bidirectionnel
- ✓ Bus d'adresse : Il contient l'adresse de la case mémoire concernée par la lecture ou l'écriture. C'est une entrée
- ✓ Bus de commandes :
  - AS : (Address Strobe) : validation du Bus d'adresse. C'est une entrée
  - DTACK (Data Acknowledge) : validation du bus dus de données. C'est une sortie.
  - CS (Chip Select) ou CE (Chip Enable) : sélection de boitier. C'est une entrée.
  - $\overline{R}/W$ $R/\overline{W}$ (Read/Write) : Signal de lecture/écriture. C'est une entrée.

3. Caractéristiques

Une mémoire est caractérisée par les informations suivantes :

- ✓ Largeur d'une mémoire : C'est la taille d'un mot, c'est-à-dire la taille d'une information transférable en un seul cycle de lecture/écriture. La largeur est donc égale à la taille du bus de donnée.
- ✓ Profondeur d'une mémoire : C'est le nombre de mots qu'on peut stocker dans la mémoire. Elle définit la taille du bus d'@. Par exemple, si le bus d'adresses comprend k fils, la profondeur sera de 2^k (= nombre d'adresses différentes qu'on peut générer avec k bits)

Rq : On parle souvent de largeur d'un bus pour définir le nombre de fils du bus.

- ✓ Capacité : Largeur * profondeur.

Le tableau suivant définis les multiples des unités utilisées pour définir les grandeurs relatives aux mémoires.

Symbole	Préfixe	Correspondance
K	Kilo(binaire)	2^{10}
M	Méga(binaire)	2^{20}
G	Giga(binaire)	2^{30}
T	Tera(binaire)	2^{40}

4. Constitution de plans mémoire

On a parfois besoin de constituer des mémoires avec des largeurs et des profondeurs différentes de celles des boitiers à notre disposition, pour les rendre, par exemple, compatibles avec les microprocesseurs.

Exemple : un microprocesseur 16 bits (taille du bus de données) utilise des mémoires de 16 kmots.
On a à notre disposition des blocs mémoire de capacité égale à 16 Ko.

Deux cas de figure principaux se présentent alors :

- ✓ Soit on dispose de mémoires de largeur 8 bits.

	µP	Mem.
Taille bus @	14	14
Taille bus données	16	8

La mémoire et le processeur sont non compatibles en largeur. On va donc associer des mémoires de façon à les rendre compatible en largeur en réalisant un montage parallèle.

- ✓ Soit on dispose de mémoires de largeur 16 bits.

	µP	Mem.
Taille bus @	14	13
Taille bus données	16	16

La mémoire et le processeur sont non compatibles en profondeur. On va donc associer des mémoires de façon à les rendre compatible en profondeur en réalisant un montage série.

Si les mémoires et le processeur sont non compatibles en largeur ET en profondeur, on réalisera un montage série/parallèle.

a. Association en parallèle

L'objectif de cette association est d'augmenter la largeur de la mémoire.

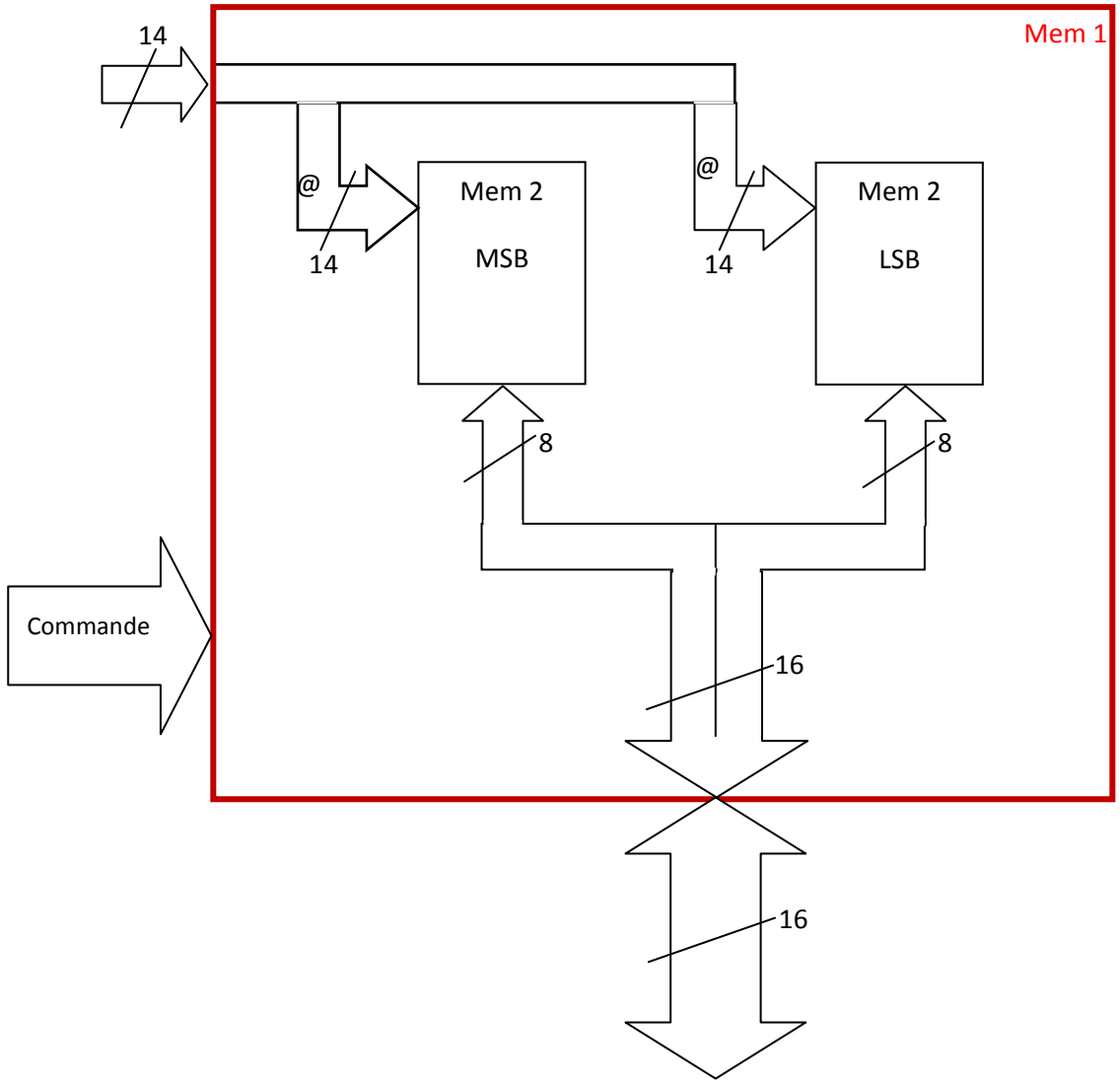
Reprenons l'exemple précédent : on veut construire une mémoire - appelons-la Mem 1 - de largeur égale à 16 bits et de profondeur égale à 16kmots

On dispose de boitiers – appelons-les Mem 2 – tels que :

- ✓ largeur = 8 bits
- ✓ profondeur = 16 Kmots

Nous devons doubler la largeur des boitiers Mem2. On utilisera donc deux boitiers de ce type de mémoire.

Dans ce cas, les infos sur le bus de commande sont les même pour toutes les mémoires.



b. Association en série :

L'objectif de cette association est d'augmenter la profondeur de la mémoire.

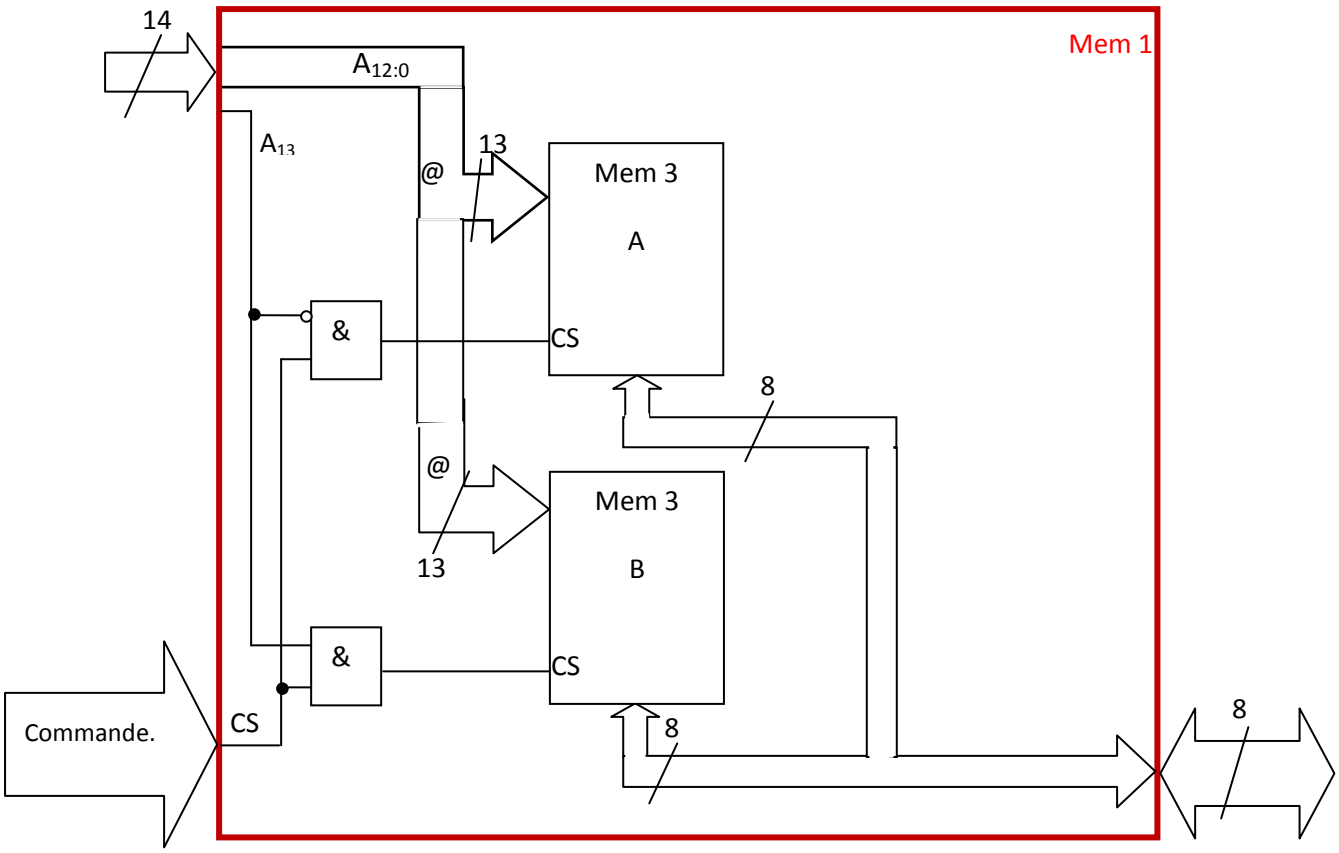
Reprenons l'exemple précédent : on veut construire une mémoire - appelons-la Mem 1 - de largeur égale à 16 bits et de profondeur égale à 16kmots

On dispose de boitiers – appelons-les Mem 3 – tels que :

- ✓ largeur = 16 bits
- ✓ profondeur = 8 Kmots

Nous devons doubler la profondeur des boitiers Mem 3. On utilisera donc deux boitiers de ce type de mémoire.

Les infos du bus de commande (sauf CS) sont les mêmes pour toutes les mémoires.



III. L'adressage des mémoires et périphériques.

1. Introduction

Dans les systèmes à µp, l'espace mémoire adressable est partagé en plusieurs zones qui permettent la localisation, donc la sélection exclusive de circuits mémoires ou de circuits périphériques.

Tous ces circuits doivent être accessibles à des adresses prédéfinies sans qu'il y ait recouvrement de zone, ce qui entrainerait des conflits d'accès.

Pour réaliser cela, on procède à des techniques de décodage du bus d'adresse qui s'opèrent en deux phases :

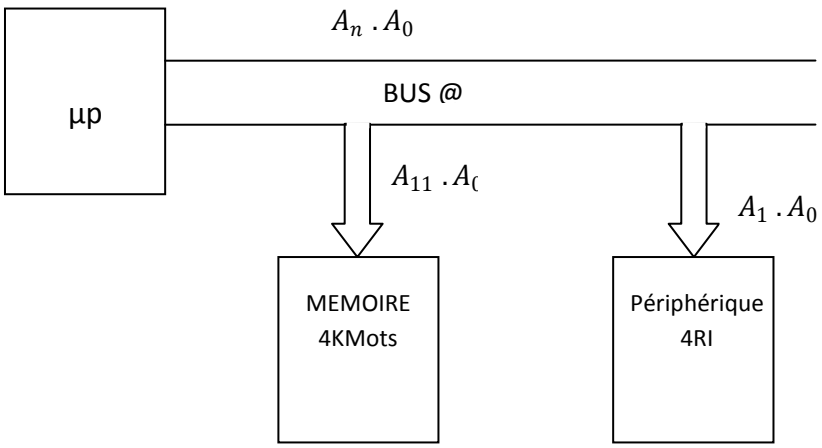
- ✓ La sélection des adresses physique
- ✓ La sélection des adresses de base

2. Décodage d'adresse : Sélection des adresses physiques :

Une mémoire de 2^n mots ou un circuit périphérique de 2^n registres adressables nécessitent n fils d'adresse pour la sélection de l'un de ses mots ou registres. Ces n fils d'entrée sont simplement réunis aux n bits de plus faible poids du bus adresse.

Ainsi, une mémoire de 4Kmots (soit 2^{12}) nécessite 12 fils d'entrée réunis aux bits A_{11} à A_0 du bus d'adresse. Un circuit périphérique de 4 registres internes (RI) (2^2) nécessite 2 fils d'entrée réunis aux bits $A_1 . A_0$ du bus d'adresse (d'un µp).

Nous obtenons le schéma suivant :



Cette première phase d'adressage s'effectue, sauf cas particuliers, toujours de la même façon.

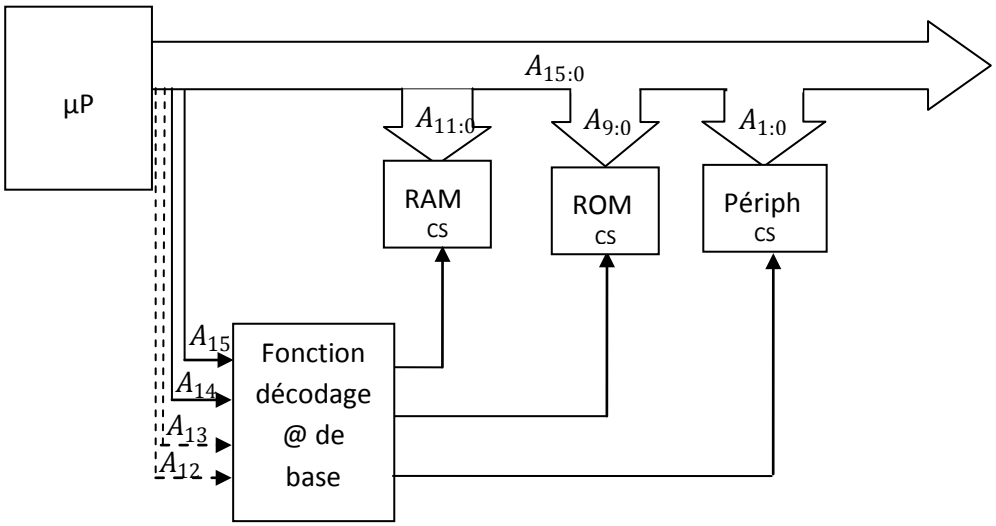
3. Décodage d'adresse : Sélection de l'adresse de base :

Si un système à µp contient plusieurs mémoires et périphériques, on doit procéder à la sélection d'un seul circuit à chaque accès (affectation exclusive). C'est la phase dite de "décodage adresse".

Tout circuit électronique adressable dispose de signaux d'entrée de validation de boitier. Ce sont ces signaux (CS ou CE) qui seront exploités par la fonction de décodage de l'adresse de base d'un circuit.

Cette sélection s'effectue par la gestion de tout ou partie des bits du bus d'adresse (bits de poids fort) qui ne sont pas utilisés par la 1ère phase d'adressage (adressage physique).

En reprenant notre exemple précédent, on peut s'attendre à l'organisation suivante : (bus @ sur 16bits)



Pour réaliser la fonction de "décodage adresse de base", on a recours à deux modes de sélection d'adresses :

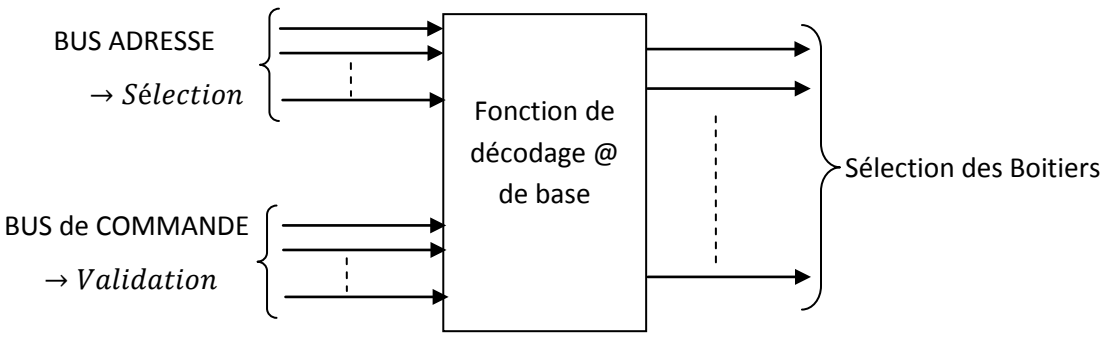
- ✓ L'adressage par sélection linéaire
- ✓ L'adressage par sélection de zones.

Ces deux modes se combinent souvent entre eux et entrainent à des équations logiques ± complexes mises en pratique par des techniques de logique câblée ou programmée (circuits programmables).

Remarque :

Pour qu'un décodage adresse soit opérationnel, il faut impérativement rajouter au mécanisme de décodage des adresses, des signaux de validations de la fonction de décodage. Ces signaux de condition valident l'échantillonnage du bus adresse et le sens des transferts (ex : AS_RD_WR), et parfois, inhibent la fonction de décodage pour des cycles particuliers du processeur (ex: i_t, Z_{bus},..).

Nous obtenons le schéma suivant :



Certains processeurs offrent la possibilité d'accéder directement à des circuits périphériques. Dans cette structure, les transferts d'Entrée/Sorties sont implémentés par des instructions spécifiques aux périphériques et font intervenir des signaux particuliers pour les échanges.

Dans ce cas, le décodage d'adresses devra intégrer la gestion de ces signaux (par exemple : signaux VMA et VPA du 68K)

a. Adressage par sélection linéaire.

Dans ce mode, la méthode consiste à relier des lignes d'adresses individuelles, aux boîtiers mémoires ou périphériques.

Exemple sur un bus d'adresses de 16 bits:

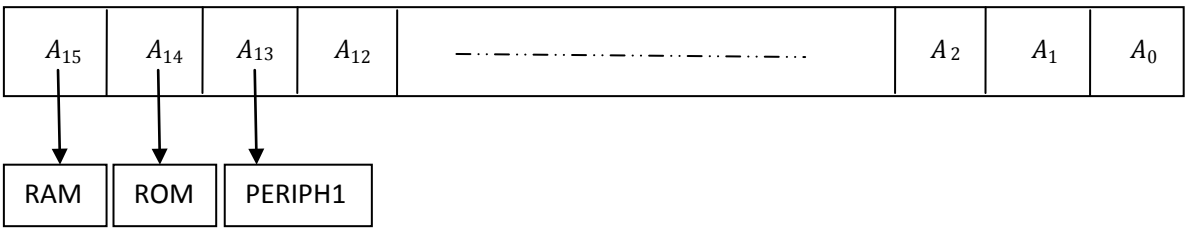
Soit un système utilisant une RAM de 2Kmots, une ROM de 1Kmots et un circuit d'Entrée/Sortie "PERIPH1" utilisant 16 mots. Nous allons effectuer, arbitrairement, la sélection suivant :

Sélection des @ physiques: RAM 2KM ⇒ 11 fils sont nécessaires (@A_{10} . A_0)
ROM 1KM ⇒ 10 fils sont nécessaires (@A_9 . A_0)
PERIPH1 16Mots ⇒ 4 fils sont nécessaires (@A_3 . A_0)

Il nous reste alors de disponible pour la sélection des adresses de base:

- A_{15} → Séléction RAM : $CS_{RAM} = AS.A_{15}$
- A_{14} → Sélection ROM : $CS_{rom} = AS.A_{14}$
- A_{13} → Sélection PERIPH : $CS_{perip\ h} = AS.A_{13}$
- A_{12} (*non utilisé)*
- A_{11} (*non utilisé)*

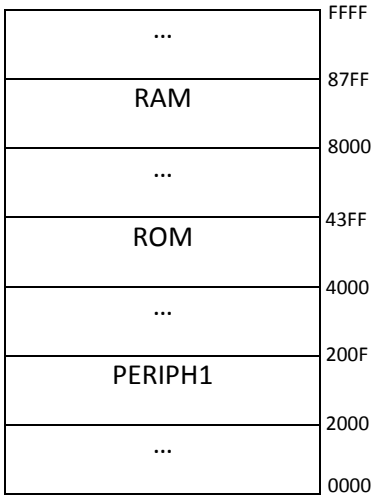
On obtient la sélection suivante :



Exploitation de l'espace mémoire :

	@ de sélection	
RAM	@ 8000 à 87FF	@ 8000 = adresse de base
ROM	@ 4000 à 43FF	@ 4000 = adresse de base
PERIPH1	@ 2000 à 200F	@ 2000 = adresse de base

Représentation de l'espace mémoire.



Avantages de ce mode :

Simplicité, pas de logique spéciale.

Inconvénients de ce mode :

- ✓ Aucune protection contre les conflits d'accès (ex: adressage de la RAM en C000). Si on veut éviter ces conflits, il faudrait rajouter des opérateurs logiques, ce qui ajoute de la complexité au mode (ex : RAM : A_{15} = 1, A_{14} = 0 et A_{13} = 0)
- ✓ Exploitation limitée de l'espace mémoire
- ✓ Evolution du système très limitée (pour rajouter des circuits supplémentaires).

CI : Ce mode convient parfaitement pour des petits systèmes à base de µp (système dédié).

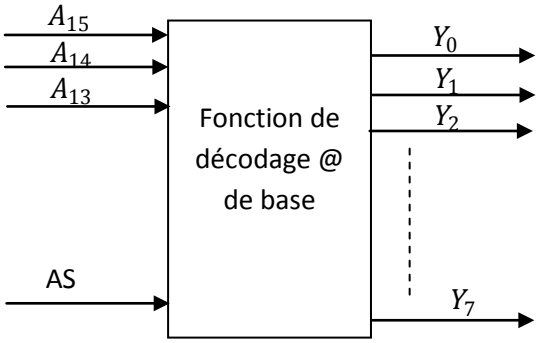
b. Adressage par sélection de zones.

Dans ce mode, on découpe l'espace mémoire en zones (ou blocs). Chaque zone se caractérise par une profondeur mémoire fixe (appelée BANK).

Le principe prend en compte deux éléments : la zone maximale adressable et le nombre de zones défini dans l'espace mémoire. La méthode consiste à prendre les n bits de poids fort du bus adresse qui permettent de créer les 2^n zones sélectables.

Exemple : Soit un bus d'adresse de 16 bits. On souhaite découper l'espace adressable en blocs de 8Kmots.

- ✓ Adressage physique : 8Kmots = 2^{13}, soit 13 fils reliés aux 13 bits de poids faible du bus d'adresse ($A_{12:0}$)
- ✓ Il reste donc 3 fils pour l'adressage de base. On pourra donc créer 8 zones de 8Kmots chacune.



Remarques :

- ✓ Risques de redondance pour les blocs partiellement exploités (sinon place sur Y2 une mémoire de 4K, elle est accessible de 2000 à 2FFF et de 3000 à 3FFF).
- ✓ Possibilité de construire une mémoire contigüe qui exploite l'ensemble de l'espace adressable.

Chap.4. Microprocesseur 68000

Document lié : Annexe 1 – Documentation Motorola 68000

I. Structure d'un microprocesseur

Un microprocesseur comprend les éléments suivants :

- ✓ Une unité de calcul
- ✓ Une unité de décodage et de séquencement des instructions
- ✓ Des unités "mémoire"
- ✓ Des compteurs

1. L'unité de calcul : Unité Arithmétique et Logique (UAL ou ALU)

C'est l'organe capable de faire des calculs arithmétiques et logiques. La réponse fournie par l'ALU comprend non seulement la valeur du résultat, mais aussi des indicateurs de condition (les flags).

2. L'unité de décodage et de séquencement : le séquenceur

C'est l'organe chargé de décoder les instructions pour déterminer l'opération à réaliser et l'adresse des opérandes.

Le séquenceur peut être câblé (tout en matériel) ou microprogrammé. L'avantage de cette seconde solution réside principalement dans le fait qu'on peut alors changer le jeu d'instructions uniquement en changeant le microprogramme.

3. Les unités "mémoire" : les Registres

Il existe deux types de registres :

- ✓ Registres non adressables : Ils sont non accessibles par l'utilisateur et à usage interne au microprocesseur. Les deux registres non adressables principaux sont le registre d'instruction (IR), qui contient l'instruction en cours d'exécution et le registre d'état (SR) qui contient l'état du microprocesseur à chaque instant.

- ✓ Registres adressables : Ils sont accessibles par l'utilisateur en lecture et en écriture. On les appelle aussi Registres Généraux.

  Ce sont des "cases mémoire" internes au processeur, ils sont donc d'accès rapide. On privilégiera leur utilisation autant que possible.

4. Les compteurs

Ce sont des registres ayant la capacité de s'incrémenter ou de se décrémenter automatiquement. On trouve généralement 2 compteurs dans les processeurs : le compteur de programme, ou compteur ordinal (PC) qui contient l'adresse de la prochaine instruction, et le pointeur de pile (SP) qui contient l'adresse du sommet de la pile.

II. Le microprocesseur 68000 : Structure et généralités

Pour pouvoir programmer convenablement un microprocesseur, l'utilisateur a besoin de connaître un certain nombre de caractéristiques spécifiques au composant, telles que :

- ✓ La définition des registres généraux (nombre et taille)
- ✓ Le jeu d'instruction
- ✓ La définition des flags
- ✓ Les types de données utilisées
- ✓ La définition des registres
- ✓ Les modes d'adressages
- ✓ La gestion de la mémoire et des registres
- ✓ La taille des bus

Le 68000 est un microprocesseur 16 bits, ce qui signifie que le bus de données est sur 16 bits. Le bus d'adresse comprend, quant à lui, 24 fils.

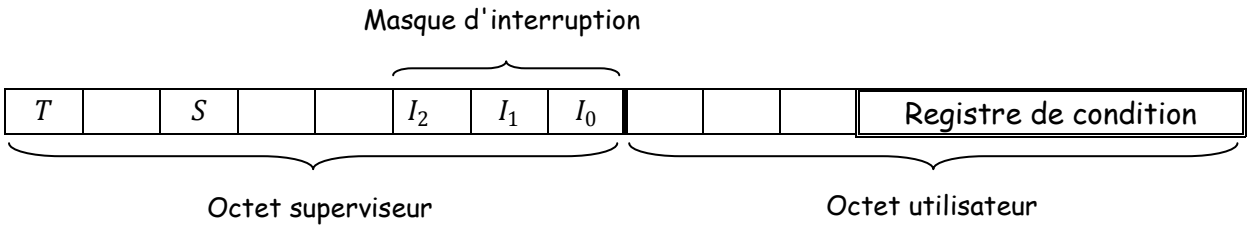
1. Les registres

a. Registres généraux

Le 68000 possède 16 registres généraux : 8 registres de données, notés D0 à D7 et 8 registres d'adresse, noté A0 à A7. Chaque registre contient 32 bits.

b. Registre d'état

Le registre d'état du 68000 est le seul registre codé sur 16 bits du 68000. Sa structure est donnée par la figure suivante :



T : Bit Trace (voir chapitre Exception)

S : Définit le mode de fonctionnement du processeur. S'il est à 1, on travaille en mode superviseur, sinon, on est en mode utilisateur.

$I_2 I_1 I_0$: Définit le masque d'interruption (voir chapitre Exception)

Registre de condition : Contient les flags

2. Les compteurs

Le 68000 comprend 2 compteurs :

- ✓ PC : Compteur ordinal
- ✓ SP : Pointeur de pile. Pour le 68000, il y a deux pointeurs de pile : le pointeur "utilisateur", qui est, par défaut est le registre d'adresses A7, et un pointeur "système" ou "superviseur" qui est un registre non adressable A7'. La sélection du pointeur SP se fait automatiquement en fonction de l'état du bit S du registre d'état.

3. Le jeu d'instruction

Il comprend environ 80 instructions. Il est disponible dans la documentation du composant.

Le format général d'une instruction est le suivant :

Opération.type_de_donnée @_source,@destination

4. Le type de données

Le 68000 peut gérer les 5 types de données suivantes :

- ✓ Octet (.B) sur 8 bits
- ✓ Mot (.W) sur 16 bits
- ✓ Double Mot (.L) sur 32 bits
- ✓ Bit
- ✓ BCD

Pour les opérations sur les bits ou sur les données codées en BCD, il existe des instructions spécifiques.

5. Le registre de conditions

Le 68000 comprend 5 flags, XZNCV, qui se trouvent dans les 5 bits de poids faible du registre d'état. On y trouve :

- ✓ Flag Z : Indique si le résultat est nul
- ✓ Flag N : Indique si le résultat est négatif
- ✓ Flag C : Indique si l'opération a généré une retenue (arithmétique non signée)
- ✓ Flag V : Indique si l'opération a généré un dépassement de capacité (arithmétique signée)
- ✓ Flag X : Utilisé pour le travail en mode étendu, c'est-à-dire sur des opérations traitant de nombres de taille supérieure à 32 bits.

Les autres éléments dont la connaissance est nécessaire à la programmation du processeur seront abordés dans les parties suivantes.

III. Utilisation des registres et de la mémoire

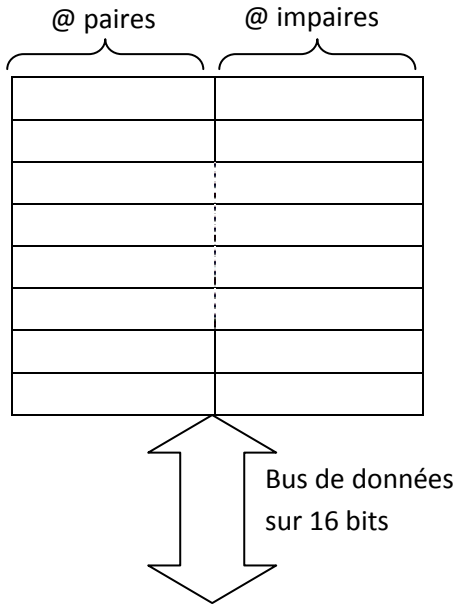
1. Les registres

Il existe plusieurs types de données, codées sur des tailles différentes. Or, tous les registres généraux sont codés sur 32 bits. La règle veut que les données occupent les poids faibles de ces registres. Ainsi, une donnée de type octet se trouvera dans l'octet de poids faible du registre général, une donnée de type mot occupera les deux octets de poids faibles du registre. Quant aux données de type double-mot, elles occuperont l'intégralité des registres.

2. La mémoire

Le 68000 est un processeur 16 bits. La largeur de son bus de données est donc de 16 bits. La mémoire doit donc avoir elle aussi un bus de données de 16 bits.

Or, le 68000 doit être capable de gérer des données codées sur 8 bits (octet). Il faudra donc utiliser une mémoire adressable par octet. On utilise donc 2 mémoires de largeur 8 bits, l'une correspondant aux adresses paires, l'autre, aux adresses impaires. Le schéma de principe d'une telle mémoire est donné ci-dessous.



a. Données type octet

L'octet correspondra au contenu de la case correspondant à l'adresse indiquée. Son adresse peut donc être paire ou impaire.

b. Données type mot

Un mot est formé de 2 octets. Il occupera donc 2 cases mémoire. Ces deux cases doivent se trouver sur une même ligne. De plus, l'adresse d'un mot correspond toujours à celle de son octet de poids fort.

⇒ L'adresse d'un mot est toujours paire

c. Données type double-mot

Un double-mot est formé de 2 mots. Il occupera donc 4 cases mémoire. Ces quatre cases occupent deux lignes. De plus, l'adresse d'un double-mot correspond toujours à celle de son octet de poids fort.

⇒ L'adresse d'un double-mot est toujours paire

IV. Exécution d'une instruction

Une instruction contient les informations suivantes :

- ✓ L'opération à réaliser
- ✓ Les adresses des opérandes
- ✓ Le type des données manipulées

L'exécution d'une instruction se passe en deux cycles consécutifs :

- ✓ une phase de recherche (fetch)
- ✓ une phase d'exécution (execute)

C'est le séquenceur qui gère ces deux phases.

1. Phase de recherche

L'instruction dont l'adresse se trouve dans le registre PC est routée de la mémoire vers le registre d'instruction IR. Elle est alors décodée par le séquenceur qui en extrait les informations utiles (opération, adresses des opérandes, type de données).

Après ce décodage, le PC est incrémenté pour pointer sur l'instruction suivante.

2. Phase d'exécution

Le séquenceur envoie les commandes nécessaires à l'ALU pour qu'elle réalise l'opération demandée. Il procède aussi au routage des données vers l'unité de calcul.

L'ALU exécute l'opération demandée sur les données et retourne le résultat (valeur + flags). La valeur est sauvegardée et le registre de conditions est mis à jour.

Rq : L'enchainement de ces deux phases est automatique.

3. Rupture de séquence.

On parle de Rupture de séquence lorsque l'instruction suivante ne se trouve pas immédiatement après l'instruction en cours.

Exemple : Structure conditionnelles : appels sous-programme

On utilisera alors des instructions qui modifient le contenu de PC.

4. Le jeu d'instruction.

La puissance d'un jeu d'instructions dépend :

- ✓ De sa richesse, c'est-à-dire du nombre d'instructions
  - CISC = Complex Instruction Set Computer
  - RISC = Reduced Instruction Set Computer
- ✓ De sa diversité c'est-à-dire du nombre d'opérations différentes.
- ✓ Des types de données utilisables.
- ✓ Des modes d'adressage.

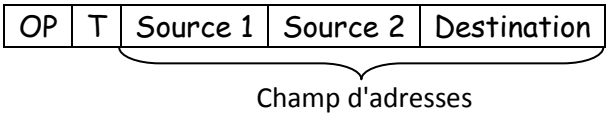
Structure générale d'une instruction :

OP	T	Source 1	Source 2	Destination

Les instruction transitent de la mémoire vers le µP via le bus de données. La taille de l'instruction doit être un multiple de la taille du bus de données. De plus, Les instructions les plus utilisées doivent être les plus courtes.

V. Techniques d'adressage

La structure générale d'une instruction en mémoire est donnée ci-dessous :



Le champ OP représente le code opération. Il occupe de 5 à 8 bits.

Le champ T représente le type de données. Il occupe de 2 à 3 bits.

Le champ d'adresses contient les adresses des différentes opérandes. Comme le bus d'adresse du 68000 contient 24 fils, ce champ contient 3x24, soit 72 bits.

Si on laissait le codage sous cette forme, chaque instruction occuperait au maximum 83 bits. Comme on peut lire 16 bits à chaque cycle de lecture (taille du bus de données), l'exécution d'une instruction prendrait beaucoup trop de temps. Il faut donc essayer de réduire la taille des instructions. Pour cela, on va diminuer la taille du champ d'adresses (le plus volumineux).

1. Réduction du nombre d'opérandes

On va réduire le nombre d'opérandes en passant de 3 à 2. Pour cela, on accepte qu'une des deux sources fasse aussi office de destination.

Ainsi, on n'aura plus : $c \leftarrow a + b$, mais, $b \leftarrow a + b$.

2. Espace d'adressage

On sait que les données peuvent se trouver soit dans un registre général du processeur, soit dans la mémoire, soit dans un registre interne d'un périphérique.

a. Espace Unique

Si on utilise un espace d'adressage unique, seule l'adresse différenciera les registres généraux, les registres des périphériques et la mémoire. L'avantage d'une telle méthode, c'est que la façon de préciser l'adresse est la même quelque soit l'endroit où se trouve la donnée. L'inconvénient, c'est que l'adresse sera toujours longue, quelque soit l'endroit où elle se trouve!

b. Espace multiple

On a vu que les données pouvaient se trouver dans 3 espaces différents :

- ✓ L'espace des registres généraux
- ✓ L'espace des registres des périphériques
- ✓ L'espace mémoire

Il suffira alors de préciser l'espace dans lequel l'adresse doit être interprétée.

On va chercher à profiter ici du nombre limité de registres du µP

Pour le 68000, on dispose de 16 registres généraux : 8 de données et 8 d'adresses. Si on sait que la donnée se trouve dans un registre du processeur, et qu'on sait aussi dans quel type de registre il se trouve, 3 bits seulement seront néecssaires pour la localiser.

Si la donnée se trouve dans un registre général, on aura une adresse courte, et donc, une instruction courte.

Rq : L'utilisation optimale d'un processeur passe par l'utilisation intensive des registres généraux. En effet, ces registres sont rapides d'accès, car internes au processeur, et les instructions qui les utilisent sont courtes, donc rapidement interprétables.

3. Modes d'adressage

Codage d'une instruction (68000)

OP	T	Ref. d'@ source	Ref d'@ destination	Instruction de base
0, 1 ou 2 mots pour préciser une valeur immédiate				Mots d'extension
0, 1 ou 2 mots d'extension pour l'adresse source				
0, 1 ou 2 mots d'extension pour l'adresse de destination				

Une instruction comprend :

- ✓ Une instruction de base qui est codée sur 1 mot
- ✓ Eventuellement, si nécessaire, de 1 à 4 mots d'extension qui servent à apporter des précisions sur la donnée (valeur ou adresse)

Dans l'instruction de base, les champs "Ref. d'@" sont chacun codés sur 6 bits :

- ✓ 3 bits pour préciser le mode d'adressage
- ✓ 3 bits pour spécifier soit le numéro du registre général utilisé, soit le numéro de sous-mode.

Le 68000 possède 13 modes d'adressage.

➢ Adressages directs

Dans ce type d'adressage, la donnée se trouve directement dans le registre indiqué dans l'instruction. On distingue :

a. L'adressage direct par registre de données

Rq : C'est le seul mode d'adressage qui concerne les registres de données

Syntaxe : Dn, où n est le numéro du registre

Ex : MOVE.B D0,D1 : L'octet de poids faible contenu dans D0 est copié dans l'octet de poids faible de D1.

b. L'adressage direct par registre d'adresse

Syntaxe : An, où n est le numéro du registre

Ex : MOVE.L D0,A1 : Le contenu de D0 est copié dans A1.

- Adressages indirects

Dans ce type d'adressage, le registre d'adresse joue le rôle de pointeur sur la donnée. On reconnait les modes d'adressage indirects aux parenthèses présentes dans la syntaxe. On distingue :

c. L'adressage indirect par registre d'adresse

Syntaxe : **(**An**)**, où n est le numéro du registre

Principe :

- ✓ L'adresse de la donnée se trouve dans le registre An (An = pointeur)
- ✓ C'est l'adresse d'un registre qui est codée dans l'instruction de base (instruction courte)

Ex : MOVE.L D0,(A1) : Le contenu de D0 est copié dans la case mémoire dont l'adresse est contnue dans A1.

d. L'adressage indirect avec post-incrémentation

Syntaxe : **(**An**)+**, où n est le numéro du registre

Principe :

- ✓ L'adresse de la donnée se trouve dans le registre An (An = pointeur)
- ✓ C'est l'adresse d'un registre qui est codée dans l'instruction de base (instruction courte)
- ✓ Le contenu du registre An est ensuite incrémenté de façon à ce que An pointe ensuite sur la donnée suivante. On incrémente donc de :
  - o +1 si on travaille en .B
  - o +2 si on travaille en .W
  - o +4 si on travaille en .L

Ex : MOVE.L D0,(A1)+ : Le contenu de D0 est copié dans la case mémoire dont l'adresse est contnue dans A1. Puis, A1 est incrémenté de 4.

Rq : A1 est modifié à l'issue de l'exécution de l'instruction

e. L'adressage indirect avec pré-décrémentation

Syntaxe : **-(**An**)**, où n est le numéro du registre

Principe :

- ✓ Le contenu du registre An est d'abord incrémenté de façon à ce que An pointe ensuite sur la donnée précédente. On décrémente donc de :
  - -1 si on travaille en .B
  - -2 si on travaille en .W
  - -4 si on travaille en .L
- ✓ L'adresse de la donnée se trouve dans le registre An décrémenté (An = pointeur)
- ✓ C'est l'adresse d'un registre qui est codée dans l'instruction de base (instruction courte)

Ex : MOVE.W D0,-(A1) : A1 est décrémenté de 2. Puis, les deux octets de poids faible de D0 sont copiés dans la case mémoire dont l'adresse est contnue dans A1.

Rq : A1 est modifié à l'issue de l'exécution de l'instruction.

f. L'adressage indirect avec base et déplacement

Syntaxe : $\mathbf{d_{16}}$**(**An**)**, où n est le numéro du registre, et d_{16} représente le déplacement, codé sur 16 bits. Il peut être indiqué en décimal, en binaire (%d_{16}) ou en héxadécimal (\$$d_{16}$)

Principe :

- ✓ L'adresse effective de la donnée est égale à An+d_{16}.

Tous les calculs d'adresse se font en arithmétique signée, sur 32 bits. Il faudra donc amener d_{16} sur 32 bits par extension de signe.

- ✓ C'est l'adresse d'un registre qui est codée dans l'instruction de base (instruction courte)
- ✓ Il faut préciser, en plus du registre de base, la valeur du déplacement. Ce mode d'adressage nécessite donc 1 mot d'extension.

Ex : MOVE.W -(A4), \$84(A3): On donne : A4 = \$0013443A
A3 = \$00138412

Détermination des adresses effectives :

- ✓ Adresse source :A4 – 4 = $00134436 = A4
- ✓ Adresse destination : $ 00138412
  + $ 00000084
  $ 00138496

Rq :Le contenu de A3 n'est pas modifié.

g. L'adressage indirect indexé avec base et déplacement

Syntaxe : **d_8(**An,Rm.Y**)**, où : An est le registre de base,
d_8 représente le déplacement, codé sur 8 bits. Il peut être indiqué en décimal, en binaire (%d_8) ou en héxadécimal ($$d_8$)
Rm.Y est le registre d'index. Il peut s'agir d'un registre d'adresse ou de données, et Y indique si on considère seulement les 16 bits de poids faible (.W) ou l'intégralité du registre (.L).

Principe :

- ✓ L'adresse effective de la donnée est égale à An+d_8+ Rm.Y.

Tous les calculs d'adresse se font en arithmétique signée, sur 32 bits. Il faudra donc amener d_8 et Rm.W sur 32 bits par extension de signe.

- ✓ C'est l'adresse d'un registre qui est codée dans l'instruction de base (instruction courte)
- ✓ Il faut préciser, en plus du registre de base, la valeur du déplacement et les informations concernant le registre d'index. Ce mode d'adressage nécessite donc 1 mot d'extension dont la structure est donnés ci-dessous :

D/A	N° Reg d'index (3 bits)	W/L	0	0	0	Déplacement sur 8 bits

D/A = sert à indiquer si Ri est un registre d'@ ou de données
= 1 si R = A
= 0 si R = D
W/L = 1 si Rm.**L**
= 0 si Rm.**W**

Ex : MOVE.W %11010 (A4, D2.W), 66(A3). On donne : A4 = $00132410
D2 = $ACDFAC24
A3 = $0013B330

Détermination des adresses effectives :

- ✓ Adresse source :

  $00132410
  + $0000001A (Déplacement)
  + $FFFFAC24 (Registre d'index amené sur 32 bits par extension de signe)
  $0012D04E

- ✓ Adresse destination : $ 0013B330
  + $ 00000042
  $ 0013B372

Rq :Les contenus de A4 et de A3 ne sont pas modifiés.

- ➢ Autres modes d'adressage

h. Adressage absolu court

Syntaxe : @.W où @ représente la valeur de l'adresse effective codée sur 16 bits.

Rq : On a besoin d'un mot d'extension pour préciser la valeur de l'adresse

Ex : MOVE.W $1234.W, 66(A3).

i. Adressage absolu long

Syntaxe : @(.L) où @ représente la valeur de l'adresse effective codée sur 32 bits.

Rq1 : On a besoin de deux mots d'extension pour préciser la valeur de l'adresse

Rq2 : Le .L est facultatif

Ex : MOVE.W $12345678, 66(A3).

j. Adressage relatif

Syntaxe : EA(PC)

Rq : EA représente l'adresse effective de la donnée, mais, c'est le déplacement relatif entre la donnée et l'adresse de l'instruction qui est codé, sur 16 bits dans le mot d'extension.

Ex : MOVE.W $1234(PC), 66(A3).

k. Adressage relatif indexé

Syntaxe : EA(PC,Rm.Y)

Rq : L'adresse effective est calculée en additionnant EA+Rm.Y, mais, c'est le déplacement relatif entre EA et l'adresse de l'instruction qui est codé, sur 8 bits dans le mot d'extension. Rm.Y représente le registre d'index (cf. adressage indirect indexé avec base et déplacement)

Ex : MOVE.W $14(PC,D0.L), 66(A3).

l. Adressage immédiat

Syntaxe : #data où data représente la valeur de la donnée. Elle peut être précisée en décimal (#42), en binaire (#%10101) ou en hexadécimal (#$42)

Rq : La donnée doit être codée dans un mot d'extension, dont la taille dépend du type de données utilisé.

Ex : MOVE.W #42, 66(A3).

VI. Branchements

1. Branchements inconditionnels

Deux instructions permmettent de réaliser un branchement inconditionnel :

- ✓ BRA <label> : déplacement relatif

  PC ← PC+dn ⇒ Code relogeable

- ✓ JMP <label> → déplacement absolu

  PC ← <label> ⇒ Code non relogeable

Rq : structure d'un programme

Etiquette Opération.Typedonnés Champ d'@ commentaire

2. Branchement conditionnels

a. Rappel : Registre de condition

Il contient les flags qui vont permettre les tests. Il correspond au poids faible du registre d'Etat (SR). On trouve :

- ✓ X → extend
- ✓ Z → zero
- ✓ N → négatif
- ✓ V → overflow (arithmétique signée)
- ✓ C → Retenue (arithmétique non signée).

b. Instruction de comparaison et de tests.

1. Instruction CMP

Elle permet de comparer 2 données.

Syntaxe : CMP.<format> <source>, <destination>

Les flags sont mis à jour en fonction du résultat de l'opération suivante : <destination> - <source>. Le valeur de cette différence n'est pas sauvegardée.

Remarque : La destination doit être un registre de données. = Dn

Variantes

- ✓ -CMPA <destination> = An
- ✓ CMPI <source> = Valeur immédiate.
- ✓ CMPN <source> = $(a_y)+$, <destination> =$(a_x)+$

2. Instructions de test

- ✓ **TST**

Elle permet de comparer une donnée à 0. A l'issue de cette instruction, seuls les flags sont mis à jour.

Syntaxe : TST.<format> <e.a>

Variante

- ✓ TAS : Test And Set (que pr les octets)

- ✓ **BTST**

Elle permet de comparer 1 bit d'une donnée à 0

Syntaxe : BTST.<format> #data,<e.a>

BTST.<format> Dn,<ea>

$\left|\begin{matrix}\#\ data \\ Dn\end{matrix}\right|$ permettent de préciser le numéro (la position) du bit à tester, sachant que cette valeur est précisée modulo 32 et que le bit de poids le plus faible correspond au numéro 0.

Variantes :

- ✓ BCLR : Teste et met le bit à 0
- ✓ BCHG : Teste et complémente le bit
- ✓ BSET : Teste et met à 1 le bit.

c. Les instructions de branchement conditionnel :

Elles permettent d'effectuer un branchement en fonction des valeurs des flags.

1. Instruction Bcc

Syntaxe : Bcc Etiquette

C'est un branchement relatif, c'est-à-dire que c'est la distance de déplacement qui est codée dans l'instruction. Selon la condition désirée, cc peut prendre plusieurs forme :

CC	pas de retenue	$\bar{C}$		
CS	retenue à 1	C		
VC	V à 0	$\bar{V}$		
VS	V à 1	V		
HI	plus grand	$\bar{C}.\bar{Z}$		binaire non signé
GE	Supérieur ou égal	$N.V + \bar{N}.\bar{V}$		binaire signé
GT	Supérieur	$NV\bar{Z} + \bar{N}.\bar{V}.\bar{Z}$		binaire signé
LS	plus petit ou égal	C + Z		binaire non signé
LE	Inférieur ou égal	$Z + N.\bar{V} + \bar{N}.V$		binaire signé
LT	inférieur	$N.\bar{V} + \bar{N}.V$		binaire signé
EQ	Egal	Z		
NE	Non Egal	$\bar{Z}$		
MI	négatif	N		
PL	positif	$\bar{N}$		

2. Instruction DBcc

Syntaxe : dBcc Dn,etiquette

Il s'agit d'une opération dite de débranchement. On commence par regarder la "condition" (i.e. le "cc" de l'instruction DBcc).

- ✓ Si elle est fausse, on décrémente le registre de 1 et on compare la valeur obtenue à -1. S'il y a égalité, on continue l'exécution du programme avec l'instruction suivante. Sinon, on se branche à l'étiquette indiquée dans l'instruction.
- ✓ Si elle est vraie, on continue l'exécution du programme avec l'instruction suivante.

3. Structures de choix !

a. Exécution conditionnelle.

```
Si <condition>
        {traitement}
Fin Si
```

```
Si      [ inst. qui positionne les flags ]
        Bcc     FinSi
        {traitement}
FinSi
```

Remarque :

- ✓ L'instruction de branchement effectue le test de la condition complémentée.
- ✓ L'étiquette FinSI doit-être externe à la structure.

Exemple : Procéder à l'échange conditionnel de D_0 et D_1 de façon à disposer du plus grand contenu D0

Type de données : Binaire Signé sur 32 bits.

```
Si D1 > D0
        {echange le contenu}
```

```
Si      CMP.L   D0, D1
        BLE     FSI
        EXG.L   D0, D1
FSI
```

b. L'alternative.

```
Si <condition>
        {traitement 1}
sinon
        {traitement 2}
fin-si
```

```
Si      [inst.positionnant les flags]
        Bcc     SINON
        {traitement 1}
        BRA FSI
SINON
        {traitement 2}
FSI
```

Remarque :

- ✓ L'instruction de branchement effectue le test de la condition complémentée.
- ✓ L'étiquette FinSI doit-être externe à la structure.

Exemple : Mettre D0 à 0 s'il est négatif et le complémenter sinon.

```
SI      TST.W   D0
        BGE     SINON
        CLR.W   D0
        BRA     FSI
SINON   NOT.W   D0
FSI
```

c. Cas multiples

```
Cas Ou <variable>
        Cas A = Traitement A
        Cas B = Traitement B
        …
        Cas N = Traitement N
        Cas Autre = Traitement  Autre
Fin Cas Ou
```

```
Cas ou
Cas A       CMP…
            BNE     Cas B
                    {traitement A}
            BRA     Fcas
Cas B       CMP…
            BNE     Cas C
                    {traitement B}
            BRA     Fcas
…
Cas N       CMP…
            BNE     CasAu
                    {traitement N}
            Bra     Fcas
Cas Autre   {traitement autre}
Fcas
```

d. Structures Itératives.

1. Boucle avec test en entrée

```
TantQue <condition>
        {traitement}
FinTantQue
```

```
Tque    [instruction qui positionne les flags]
        Bcc     FTQue
        {traitement}
        BRA     Tque
FTQue
```

Exemple : Mettre à 0 une zone mémoire pointée par A0. Le nombre de mots de cette zone est codé sur 16 bits et est contenu dans D0

```
Tque    TST.W       D0
        BEQ     Ftque
        CLR.W       (A0)+
        SUBQ.W  #1, D0
        BRA Tque
Ftque
```

2. Boucle avec test en sortie

```
Répeter
        {traitement}
Tant-que <condition>
```

```
Rep     {traitement}
        [instruction qui positionne les flags]
        Bcc     Rep
```

Exemple :Une chaine de caractères pointée par A1 et qui se termine par NULL (00). Donner le nombre de caractères de la chaine et le mettre dans D1 (on compte le caractère NULL)

```
        CLR.L       D1
Rep     ADDQ.L      #1, D1
        TST.B       (A1)+
        BNE         Rep
```

3. Boucle avec test en sortie et contrôle de limite

```
Répeter
        {traitement}
sortir si A : <condition>
sortir si B : <condition>
```

```
Rep     {traitement}
        dbcc    Dn, Rep
fin Rep
```

Exemple1 : Rechercher l'adresse du premier octet nul d'un bloc mémoire de 256 octets pointé parA1

```
        MOVE.W  #255, D1
REP     TST.B       (A1)+
        dBEQ        D1, REP
FREP    SUBA.L      #1, A1
```

Exemple2 : Traitement répétitif: on veux effectuer N fois le même traitement

```
        MOVE.W   #N-1, D0
REP     ____  }
        ____  }  Traitement
        ____  }
        DBRA     D0, REP
```

VII. Les sous programmes

1. Généralités:

La mise en oeuvre des sous programme nécéssite:

- ✓ Le transfert d'éxecution du programme appelant au sous-programme et inversement
- ✓ Le passage de paramètres (entrée/sortie)

Le point d'entrée du sous programme est fixé. La valeur peut-être implanté dans le code exécutable

L'adresse de retour est variable, elle ne peut donc pas être mémorisée dans le code exécutable, mais dans la pile (mémoire vive).

2. Appel et retour de sous programme

Deux instructions permettent d'appeler un sous –programme :

- ✓ BSR : Branch to SubRoutine (branchement relatif)
- ✓ JSR : Jump to SubRoutine (branchement absolu)

La rupture de séquence se déroule en deux étapes :

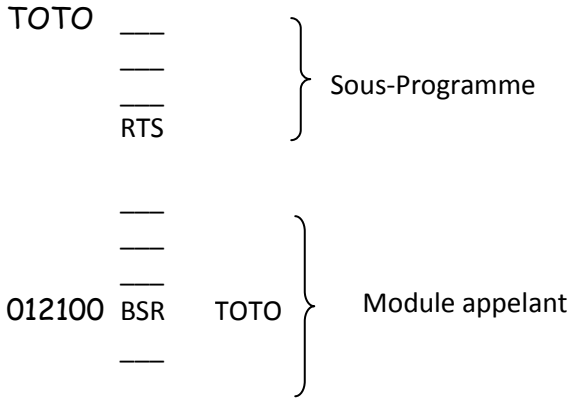
- ✓ Sauvegarde dans la pile de l'adresse de retour (avec prédécrémentation automatique de A7).
- ✓ Branchement inconditionnel

Deux instructions permettent de revenir d'un sous –programme :

- ✓ RTS : ReTurn from Subroutine. Cette instruction récupère un double mot depuis la pile et le transfère à PC (avec post-incrémentation automatique de A7).
- ✓ RTR : ReTurn and Restor condition codes. Cette instruction récupère un mot depuis la pile. Les 5 bits de poids faible de ce mot redéfinissent XNZVC. A7 est incrémenté, puis, tout se passe comme un RTS.

  Rq : Rien ne sauvegarde les flags : il faut sauvegader le registre d'état.

Exemple :



Etat de la pile

Avant bsr	Juste après bsr	Après Rts
	A7 → 0001	0001
	2104	2104
A7 →////////	//////	A7 →//////
////////	//////	//////

Remarque : A7 peut évoluer pendant l'exécution de la procédure. Il faut donc bien s'assurer que A7 pointe bien sur l'adresse de retour avant le RTS.

3. Elaboration des sous-programmes

a. Passage de paramètres :

On peut noter 3 caractéristiques principale pour chaque paramètre :

- ✓ Sens de la comunication : paramètre-entrant (appel) ou sortant (retour)
- ✓ Nature : donnée ou @
- ✓ Emplacement : registre ou mémoire ?

Plusieurs solutions sont toujours possibles. On peut cependant s'appuyer sur quelques idées générales :

- ✓ Le sens est imposé par la définition du sous programme
- ✓ La nature est liée à l'espace occupé par l'information a transmettre.
  - Passage par valeur pour les infos occupant peu de place
  - Passage par adresse pour les variables de type structuré (tableau, chaine de caractères…).
- ✓ L'emplacement de stockage dépend du nimbre et de la taille des paramètres
  - Passage par registre si le nombre et taille des paramètres sont faibles
  - Passage par la pile sinon

Exemple :Sous programme effectuant la recherche du max dans un tableau d'octets.

Paramètres entrants :

- A0 = @ du tableau (pointeur sur tableau, passé par registre).
- D4 = taille du tableau (passage par valeur et par registre).

Paramètres-sortants

- A0 = @ du max (pointeur sur max, passé par registre).

b. Localisation des variables

1. Variables globales

Un pointeur sur la zone de stockage des variables peut être utilisé par l'ensemble des modules. On peux réserver un registre d'adresses An à cet usage.

2. Variables locales

La durée de vie de ces variables étant limitée à la durée d'exécution du sous-programme, l'utilisation de la pile est donc bien adaptée à leur stockage. Il faut dans le sous-programme :

- ✓ réserver une zone de stockage
- ✓ disposer d'un pointeur sur cette zone
- ✓ libérer la zone de stockage en fin de ss-programme.

Instructions :

- ✓ LINK An, #<d_{16}>

  L'exécution de cette instruction comprend 3 étapes :

  1. sauvegarde de An dans la pile
  2. pointage sur la pile par An
  3. réservation d'une zone de pile. (d_{16}<0)

- ✓ UNLINK An

  L'exécution de cette instruction comprend 2 étapes :

  1. Libération de la zone de pile. (A7 revient sur An)
  2. Restauration du contenu de An (An est dépilé).

Exemple : Sous-programme TOTO utilisant :

- ✓ 2 paramètres-entrants utilisés par la pile :PARAM1.L ($42)

  PARAM2.W ($10)

- ✓ 3 variables locales : 1 double-mot :VAR1.L

  2 doubles mots : VAR2.W ; VAR3.W

Module appelant :

```
MOVE.L      #$42, -(A7)
MOVE .W     #$10, -(sp)
BSR         TOTO
___
___
```

Sous-programme TOTO

```
LINK        A5, #-8
___
___
UNLNK           A5
```

VIII. Les exceptions

1. Généralités

Le 68000 (comme tout microprocesseur) est doté d'un mécanisme qui permet l'apparition d'un événement fortuit, d'une situation d'exception, d'abandonner le traitement en cours et de transférer l'exécution à un autre traitement associé à cet événement.

Dans la plupart des cas, le premier traitement doit être ensuite poursuivi, ce qui implique une sauvegarde du contexte lors du transfert.

L'événement fortuit peut avoir deux origines :

- ✓ Interne au système ; il traduit une anomalie d'exécution
- ✓ Externe ; une condition materielle particulière doit être prise en compte (par exemple, un périphérique cherche à « attirer l'attention »)

Il est aussi intéressant que ce même mécanisme puisse être utilisé dans des conditions voisines d'un appel de sous-programme (en particulier pour effectuer des appels à des fonctions système). Il faut être en mesure de « programmer la situation d'exception » , la déclencher par l'exécution d'une instruction.

Plusieurs événements exceptionnels pouvant se présenter simultanément ; leur prise en compte doit être hiérarchisée. Ceci couvre l'ordre dans lequel s'effectue le lancement des traitements d'exception, mais aussi la possibilité de suspendre l'exécution de l'un d'eux au profit d'un autre considéré de priorité plus élevée.

Un microprocesseur peut distinguer un nombre donné d'exceptions de natures différents. Il fait correspondre à chacune d'elle un traitement particulier en exploitant une table d'adresses de lancement de ces traitements. La localisation de cette table est fixe, implicitement connue, mais son contenu est programmable et va être adapté à l'application développée. Ce procédé est nommé vectorisation des exceptions.

Le 68000 présente deux modes d'exécution d'un programme, le mode superviseur et le mode utilisateur. Ceci correspond à l'idée d'utiliser le système microprocesseur à 2 niveaux, un niveau superviseur pour lequel toute action est permise et un niveau utilisateur subissant quelques restrictions (jeu d'instructions réduit …). L'intérêt de cette distinction est très lié aux exceptions et permet d'améliorer la sécurité de fonctionnement des systèmes d'exploitation.

2. Exceptions d'origine interne au système.

Les paragraphes suivant décrivent les différents contextes logiciels d'exécution d'une instruction qui conduisent à un traitement d'exception.

a. Exceptions liées à une anomalie d'exécution.

Cette catégorie correspond essentiellement à des erreurs de programmation.

1. Erreur d'adressage.

Il s'agit du non respect de la contrainte d'alignement du fait d'une tentative d'accès à un mot, un mot long ou une instruction à une adresse impaire.

Le cycle bus est alors avorté et le traitement de l'exception Erreur d'adressage est enchaîné.

Si une erreur d'adressage se présente pendant une exception Erreur bus, Erreur d'adressage ou Reset, le processeur s'arrête.

2. Violation de privilège.

Un programme s'exécutant en mode utilisateur (bit S du registre SR à 0) ne doit pas faire usage des instructions privilégiées :

Andi	#<d16>,SR	Move	An,USP
Ori	#<d16>,SR	Reset	
Eori	#<d16>,SR	Stop	#<d16>
Move	<A.E.>,SR	Rte	
Move	USP,An		

Une infraction à cette règle conduit à une exception Violation de privilège.

3. Codes opération illégaux.

Le premier mot de codage d'une instruction peut prendre 2^{16} formes. L'ensemble des combinaisons possibles n'est pas effectivement utilisé par le jeu d'instructions du 68000.

Si la phase de recherche d'une instruction conduit à un premier mot différent de ceux répertoriés, une exception Instruction illégale est générée.

4. Codes opération inexistants.

Parmi les mots ne correspond pas au début d'une instruction légale, ceux dont les bits 15 à 12 son égaux à 1010 ou 1111 (mots $Axxx ou $Fxxx) forment une catégorie particulière : ils sont considérés comme des codes opération indéfinis plutôt qu'illégaux.

Deux exceptions (distinctes de l'exception Instruction illégale) leur sont associées : exception Emulateur ligne 1010 et exception Emulateur ligne 1111. Un traitement propre à chacun des deux cas est exécuté à la rencontre de ces codes.

La vocation de ces exceptions est l'émulation logicielle d'instructions nouvelles. L'utilisateur définit par l'écriture du traitement d'exception leur nature et leur effet. Les bits 11 à 0 constituent un moyen de distinguer plusieurs opérations ainsi que différents modes d'adressage.

L'usage de ces codes est donc délibéré et ne peut pas être considéré comme une erreur de programmation.

b. Exceptions programmées.

Certaines instructions ont pour conséquence de leur exécution le déclenchement d'exceptions. Ces dernières ne présentent donc pas le caractère fortuit rencontré en général.

α. Instruction TRAP.

Syntaxe : Trap #numéro

Le numéro, de valeur 0 à 15, fait partie du code de l'instruction. Il sélectionne un traitement parmi un ensemble de 16, associés à cette instruction.

La fonction habituelle de TRAP est de permettre à des programmes utilisateurs d'effectuer des appels système.

β. Instruction TRAPV.

Syntaxe : TRAPV

Le lancement du traitement de l'exception TRAPV est conditionné à la valeur de V lors de l'exécution de l'instruction :

Si V = 0, l'exécution du programme se poursuit en séquence

Si V = 1, l'exécution est transférée au traitement d'exception.

χ. Instructions DIVS, DIVU.

Une même exception est générée si l'on cherche à effectuer une division signée ou non avec un diviseur de valeur 0 (opérande origine, pointé par <A,E>),

δ. Instruction CHK.

Syntaxe : CHK Dn

L'instruction compare le contenu du registre de données Dn à l'intervalle [0,opérande] (l'opérande est défini par <A,E>).

La comparaison est effectuée en arithmétique signée.

Si (Dn)∈ intervalle, on continue le programme, sinon, exécution d'une exception CHK

c. Exception du mode trace.

Le bit T du registre SR, mis à 1, place le 68000 en mode trace.

Dans ce mode, une exception est forcée après exécution de chaque instruction.

La fonction de cette exception est une aide au développement. Le traitement correspondant est habituellement un programme de mise au point permettant de visualiser l'exécution en pas à pas.

Cas particuliers :

- ✓ Si l'instruction n'est pas exécutée parce que privilégiée, illégale, ou du fait d'une interruption, l'exception Trace ne se produit pas.

- ✓ De même, si l'exécution de l'instruction ne s'achève pas du fait d'un reset, d'une erreur bus ou d'une erreur d'adressage, l'exception Trace ne se produit pas.
- ✓ Une exception lancée par une instruction (ex : trap) est d'abord exécutée, puis est suivie de l'exception Trace.

3. Exceptions d'origine externe.

Le déclenchement d'exceptions peut aussi ne pas avoir pour origine directe le 68000, et résulter dans ce cas d'une action matérielle externe.

a. Reset

Une initialisation du système est effectuée en plaçant à l'état bas pendant au moins 100ms la broche Reset du microprocesseur (dans un même temps, la broche Halt doit également être forcée à l'état bas).

Cette action provoque une exception Reset :

- ✓ Le traitement en cours est abandonné, sans intention de le poursuivre ultérieurement (aucune sauvegarde de contexte n'est effectuée).
- ✓ Le pointeur de pile système (SSP) puis le compteur de programme (PC) sont initialisés.
- ✓ L'exécution reprend au point de démarrage du système en mode superviseur, trace inhibée et interruptions masquées.

La broche Reset est bidirectionnelle.

L'usage précédent la considère comme une entrée.

L'instruction Reset permet de commander en sortie : elle la place à l'état bas pendant 124 cycles d'horloge. Ceci ne provoque pas d'exception mais permet une initialisation programmée de périphériques.

b. Erreur bus.

Les transferts entre 68000 et mémoire ou périphériques mettent en jeu un dialogue. Si dans certaines circonstances, ce dialogue n'est pas mené à bien, le fonctionnement du système risque d'être bloqué.

Une logique externe (chien de garde, watchdog) doit alors détecter le dépassement d'un temps limite de réponse et demander au microprocesseur (par le biais d'une broche BERR) le traitement par exception de ce qui est considéré comme erreur bus.

Le cycle bus courant est avorté et l'exception Erreur Bus débute immédiatement. Une sauvegarde détaillée du contexte est effectuée.

Une erreur bus intervenant pendant une exception Erreur bus ou Reset provoque l'arrêt du processeur.

c. Interruptions.

Une demande d'interruption du traitement en cours peut émaner de périphériques. Ils doivent pour cela gérer l'état de 3 broches du microprocesseur : IPL0,IPL1,IPL2.

Le mécanisme d'interruption offre 7 niveaux de priorité (1 à 7).

La combinaison appliquée sur les entrées IPL définit le niveau de la demande :

- ✓ 7 est le niveau de plus haute priorité, 1 le plus faible.
- ✓ 0 traduit une absence de demande.

Les demandes sont détectées entre l'exécution des instructions et leur prise en compte dépend de l'état d'un masque d'interruption formé par les bits I2,I1,I0 du registre d'état :

- ✓ Les niveaux 1 à 6 sont masquables :
  - Si le niveau de la demande est supérieur à la valeur du masque, le traitement de l'interruption débute
  - Sinon, le traitement est différé jusqu'à ce que le masque devienne inférieur au niveau (le programme en cours se poursuit en séquence)
- ✓ Le niveau 7 est non masquable : Il est sensible à des fronts ; la séquence de traitement de l'exception est inconditionnellement lancée.

Le 68000 permet l'association directe de 199 traitements distincts aux demandes d'interruption, selon 2 modes :

- ✓ Par vectorisation automatique : le traitement est choisi parmi un groupe de 7, en fonction du niveau de la demande
- ✓ Par vectorisation utilisateur : le traitement est choisi parmi un ensemble de 192, par une logique externe au 68000.

4. Mécanisme.

a. Hiérarchie des priorités.

Les exceptions sont classées par groupe, à priorité décroissante entre groupes et à l'intérieur de chaque groupe :

- ✓ Groupe 0 (l'instruction en cours est abandonnée ; le lancement de l'exception débute dans les deux cycles d'horloge suivants) : reset, erreur d'adressage, erreur bus
- ✓ Groupe 1 (l'instruction est lancée à l'issue de l'instruction en cours) : trace, interruption, instruction illégale, violation de privilège.
- ✓ Groupe 2 (l'exécution de l'instruction conduit à l'exception) : Trap, TrapV, Chk, division par 0 (Rq : priorité sans objet ici)

L'exécution du traitement d'une exception est interrompue par le lancement d'une exception de plus grande priorité.

Lors de l'apparition simultanée de deux exceptions, le traitement de celle de plus faible priorité est lancé en premier (mais abandonné immédiatement au profit de celui correspondant à une priorité plus élevée).

b. Lancement d'une exception.

Le séquençage du lancement d'une exception comporte 4 phases :

- ✓ Redéfinition du contenu de SR :

1/ Le contenu du SR fait l'objet d'une sauvegarde temporaire interne au 68000

2/ Le bit S est mis à 1 (le traitement de l'exception va être exécuté en mode superviseur)

3/ Le bit T est mis à 0 (le mode trace est inactive pendant l'exception)

Ces 3 actions sont effectuées quelle que soit l'exception. Dans le cas des exceptions Reset et Interruptions, le masque de priorité d'interruption est en outre actualisé :

4/ repositionnement du masque au niveau 7 pour Reset ou au niveau de la demande pour Interruption.

✓ Détermination du numéro du vecteur d'exception :

Un vecteur est une position mémoire où le microprocesseur recherche l'adresse du traitement de l'exception.

Chaque vecteur est identifié par un numéro codé sur 8 bits (256 vecteurs peuvent être distingués). L'adresse absolue d'un vecteur est déterminée en multipliant par 4 son numéro. Ce calcul est effectué par le microprocesseur et n'est pas modifiable. L'adresse d'un vecteur donné est donc constante. Le contenu du vecteur doit être défini en fonction de l'implantation en mémoire des traitements d'exception. Il est programmé par l'utilisateur.

Le numéro est généré de façon interne et automatique par le 68000 pour toutes les exceptions, sauf les interruptions. La détermination du numéro associé à une interruption requiert un dialogue avec cette logique externe :

- le 68000 exécute un cycle de reconnaissance d'interruption au cours duquel il place sur les lignes A1,A2,A2 du bus d'adresses le niveau de l'interruption reconnue.
- une logique externe choisit une vectorisation automatique ou une vectorisation utilisateur :
    - vectorisation automatique : la logique répond par la ligne VPA, le 68000 détermine le numéro du vecteur en fonction du niveau.
    - vectorisation utilisateur : la logique délivre le numéro du vecteur sur le bus de données et termine l'échange par DTACK (un périphérique de la famille 68000 non initialisé pour cette vectorisation répond par le numéro 15, vecteur Interruption non initialisée).
- une absence de réponse par VPA ou DTACK doit être détectée par une logique d'erreur bus qui active alors la ligne BERR. Le microprocesseur associe à ce cas le vecteur numéro 24 (Interruption parasite).

La table des vecteurs occupe les adresses 0 à 1023 (0 à $3FF).

Chaque vecteur contient un mot long, excepté le vecteur Reset qui définit par 2 mots longs les contenus initiaux du pointeur de pile système et du compteur de programme.

Numéro	Adresse	Exception
0	000	Reset (SSP) initial
	004	Reset (PC) initial
2	008	Erreur bus
3	00C	Erreur d'adressage
4	010	Instruction illégale
5	014	Division par 0
6	018	Instruction CHK
7	01C	Instruction TRAPV
8	020	Violation de privilège
9	024	Trace
10	028	Emulateur ligne 1010
11	02C	Emulateur ligne 1111
12	030	Réservé
13	034	Réservé
14	038	Réservé
15	03C	Interruption non initialisée
16-23	040	Réservé
	05C	-
24	060	Interruption parasite
25	064	Auto-vecteur interruption niveau 1
26	068	Auto-vecteur interruption niveau 2
27	06C	Auto-vecteur interruption niveau 3
28	070	Auto-vecteur interruption niveau 4
29	074	Auto-vecteur interruption niveau 5
30	078	Auto-vecteur interruption niveau 6
31	07C	Auto-vecteur interruption niveau 7
32-47	080	Vecteurs instruction TRAP
	0BC	-
48-63	0C0	Réservé
	0FF	-
64-255	100 – 3FC	Vecteurs interruptions utilisateur

✓ Sauvegarde du contexte :

L'exception Reset constitue un cas particulier du fait de l'absence de sauvegarde.

Pour l'ensemble des autres exceptions, une sauvegarde de (PC) et (SR) est effectuée en pile superviseur.

Les exceptions Erreur bus et Erreur d'adressage provoquent en outre le rangement d'autres informations en pile.

- ✓ Redéfinition du contenu de PC :

Le contenu du vecteur d'exception est transféré dans le PC, puis l'exécution reprend normalement.

c. Retour d'exception.

L'instruction privilégiée RTE, exécutée en mode superviseur, transfert depuis la pile un mot vers le registre d'état SR et un mot long vers le compteur de programme PC.

Ceci entraine la reprise d'exécution du traitement interrompu, sous réserve que le pointeur de pile soit resté correctement positionné (à l'identique des conditions de retour d'un sous-programme par RTS).

En mode utilisateur, RTE conduit à une exception Violation de privilège.

5. Modes

La distinction entre les deux modes d'exécution des programmes est effectuée par le bit S du registre d'état SR :

- ✓ S = 1 mode superviseur
- ✓ S = 0 mode utilisateur

Mode superviseur :

- ✓ le niveau de privilège est le plus élevé
- ✓ toutes les instructions sont autorisées
- ✓ les cycles bus sont classés en référence superviseur
- ✓ le pointeur de pile est SSP

Mode utilisateur :

- ✓ Les instructions "privilégiées" ne peuvent pas être utilisées
  - o reset, stop, rte
  - o l'instruction de transfert avec le pointeur de pile utilisateur : move USP,An ou move An,USP
  - o -les instructions modifiant l'octet système du registre d'état SR.
- ✓ Les cycles bus sont classés en référence utilisateur
- ✓ Le pointeur de pile est USP

Remarque : En mode utilisateur, A7 est implicitement le pointeur USP. Sa modification est naturellement permise. L'accès à SSP dans ce mode est impossible.

En mode superviseur, A7 correspond à SSP, évidemment modifiable. L'instruction « move USP » a pour but d'autoriser également l'accès au pointeur utilisateur.

Changement de mode :

- ✓ passage utilisateur → superviseur : par exception exclusivement
- ✓ -passage superviseur → utilisateur : RTE,MOVE to SR , ANDI to SR, EORI to SR.

Chap.5. Introduction aux Microcontrôleurs

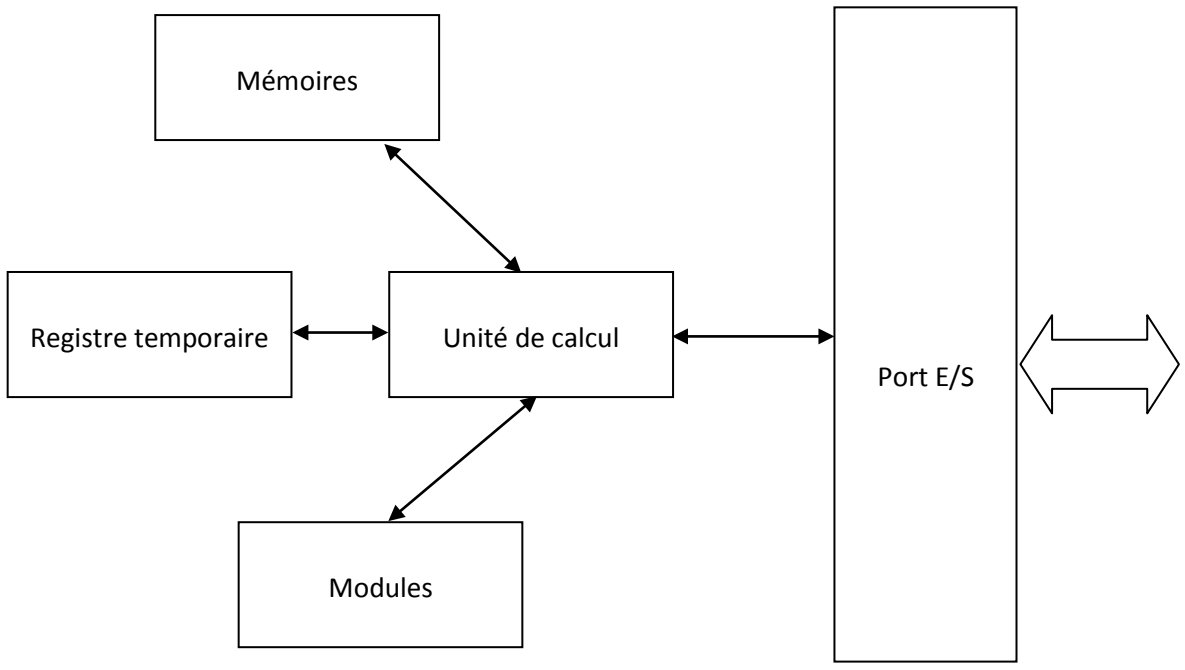
I. Généralités – Présentation du composant

Un microcontrôleur est un circuit intégré rassemblant dans un même boitier un microprocesseur, plusieurs types de mémoires et des périphériques de communication (Entrées/Sorties). La définition donnée dans l'Arrêté du 14 septembre 1990 relatif à la terminologie des composants électroniques (parue au BO) est la suivante : "Microprocesseur comprenant des éléments fixes et des éléments personalisés selon l'application".

Un microcontrôleur est donc une unité de traitement de l'information programmable permettant de réaliser des montages sans nécessiter l'ajout de composants extérieurs. Une fois programmé, il peut fonctionner de façon autonome.

Le PIC est un microcontroleur construit par MicroChip. Officiellement, ce n'est pas un acronyme, mais, la traduction généralement admise est Peripherical Interface Controller.

La structure interne (simplifiée) d'un tel composant est donné par le diagramme suivant:



Les PICS sont des composants de type RISC (Reduced Instruction Set Computer). Les instructions sont codées sur un seul mot de programme et sont, sauf pour les instructions de branchement, exécutées en un seul cycle. On différencie les PIC grâce à la taille de leurs mots d'instruction.

La dénomination des PIC donne un certain nombre d'informations :

- ✓ Le premier nombre permet de déterminer la famille (et donc la taille des mots d'instructions) du PIC
- ✓ La lettre qui suit indique le type de mémoire programme incluse dans le composant (F : Mémoire Flash ; CR : Mémoire ROM ; C = Mémoire EPROM ou EEPROM)
- ✓ Le nombre de 2 à 4 chiffres qui termine le nom indique le modèle du PIC au sein de la famille.

II. PIC 16F8X (Annexe 2)

Exercice : Déterminer les éléments caractéristiques du PIC pour pouvoir le programmer.

Objectif : Savoir lire une documentation technique